Tutorial on Sparse Fourier Transforms

Eric Price

UT Austin
The Fourier Transform
Conversion between time and frequency domains

Time Domain

Frequency Domain

Fourier Transform

Displacement of Air

Concert A
The Fourier Transform is Ubiquitous

Audio

Video

Medical Imaging

Radar

GPS

Oil Exploration
Computing the Discrete Fourier Transform

- How to compute $\hat{x} = Fx$?

Naive multiplication: $O(n^2)$.

Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

The method greatly reduces the tediousness of mechanical calculations. – Carl Friedrich Gauss, 1805

By hand: $22^n \log n$ seconds. [Danielson-Lanczos, 1942]

Can we do better?

When can we compute the Fourier Transform in sublinear time?
Computing the Discrete Fourier Transform

- How to compute $\hat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
Computing the Discrete Fourier Transform

- How to compute $\hat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

The method greatly reduces the tediousness of mechanical calculations. – Carl Friedrich Gauss, 1805

By hand: $22n \log n$ seconds. [Danielson-Lanczos, 1942]

Can we do better?

When can we compute the Fourier Transform in sublinear time?

Eric Price
Tutorial on Sparse Fourier Transforms
Computing the Discrete Fourier Transform

- How to compute $\hat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

[T]he method greatly reduces the tediousness of mechanical calculations.

– Carl Friedrich Gauss, 1805
Computing the Discrete Fourier Transform

- How to compute $\hat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

[T]he method greatly reduces the tediousness of mechanical calculations.

– Carl Friedrich Gauss, 1805

- By hand: $22n \log n$ seconds. [Danielson-Lanczos, 1942]
Computing the Discrete Fourier Transform

- How to compute $\hat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

[T]he method greatly reduces the tediousness of mechanical calculations.

– Carl Friedrich Gauss, 1805

- By hand: $22n \log n$ seconds. [Danielson-Lanczos, 1942]
- Can we do better?
Computing the Discrete Fourier Transform

- How to compute $\hat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

> The method greatly reduces the tediousness of mechanical calculations.

– Carl Friedrich Gauss, 1805

- By hand: $22n \log n$ seconds. [Danielson-Lanczos, 1942]
- Can we do much better?
Computing the Discrete Fourier Transform

- How to compute $\hat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

[T]he method greatly reduces the tediousness of mechanical calculations.

– Carl Friedrich Gauss, 1805

- By hand: $22n \log n$ seconds. [Danielson-Lanczos, 1942]
- Can we do much better?

When can we compute the Fourier Transform in sublinear time?
Idea: Leverage **Sparsity**

Often the Fourier transform is dominated by a small number of peaks:

- **Time Signal**
- **Frequency** (Exactly sparse)
- **Frequency** (Approximately sparse)
Idea: Leverage **Sparsity**

Often the Fourier transform is dominated by a small number of peaks:

- **Time Signal**
- **Frequency (Exactly sparse)**
- **Frequency (Approximately sparse)**

Sparsity is common:

- Audio
- Video
- Medical Imaging
- Radar
- GPS
- Oil Exploration
Idea: Leverage **Sparsity**

Often the Fourier transform is dominated by a small number of peaks:

- **Time Signal**
- **Frequency** (Exactly sparse)
- **Frequency** (Approximately sparse)

Sparsity is common:

**Goal of this workshop:** *sparse* Fourier transforms

*Faster* Fourier Transform on sparse data.
Classes of sparse Fourier transform algorithms

For recovering a $k$-sparse signal in $n$ dimensions.

- Exact sparsity, deterministic algorithm
Classes of sparse Fourier transform algorithms
For recovering a $k$-sparse signal in $n$ dimensions.

- Exact sparsity, deterministic algorithm
  - Vandermonde matrix: $2k$ samples sufficient
Classes of sparse Fourier transform algorithms
For recovering a $k$-sparse signal in $n$ dimensions.

- Exact sparsity, deterministic algorithm
  - Vandermonde matrix: $2k$ samples sufficient
  - Syndrome decoding for Reed-Solomon coding
Classes of sparse Fourier transform algorithms
For recovering a $k$-sparse signal in $n$ dimensions.

- Exact sparsity, deterministic algorithm
  - Vandermonde matrix: $2k$ samples sufficient
  - Syndrome decoding for Reed-Solomon coding
  - Berlekamp-Massey: $O(k^2 + k(\log \log n)^c)$ time.

Approximate sparsity, $2^{-k}$ failure probability
- Compressed sensing, using Restricted Isometry Property
- $O(k \log 4^n)$ samples, $O(n \log c n)$ time.

Today: Approximate sparsity, $1/4$ or $1/n^c$ probability.
- Using hashing
- $O(k \log c n)$ samples, $O(k \log c n)$ time.
Classes of sparse Fourier transform algorithms
For recovering a $k$-sparse signal in $n$ dimensions.

- **Exact sparsity, deterministic algorithm**
  - Vandermonde matrix: $2k$ samples sufficient
  - Syndrome decoding for Reed-Solomon coding
  - Berlekamp-Massey: $O(k^2 + k(\log \log n)^c)$ time.

- **Approximate sparsity, $2^{-k}$ failure probability**
Classes of sparse Fourier transform algorithms
For recovering a $k$-sparse signal in $n$ dimensions.

- **Exact sparsity, deterministic algorithm**
  - Vandermonde matrix: $2k$ samples sufficient
  - Syndrome decoding for Reed-Solomon coding
  - Berlekamp-Massey: $O(k^2 + k(\log \log n)^c)$ time.

- **Approximate sparsity, $2^{-k}$ failure probability**
  - Compressed sensing, using Restricted Isometry Property
Classes of sparse Fourier transform algorithms
For recovering a $k$-sparse signal in $n$ dimensions.

- Exact sparsity, deterministic algorithm
  - Vandermonde matrix: $2k$ samples sufficient
  - Syndrome decoding for Reed-Solomon coding
  - Berlekamp-Massey: $O(k^2 + k(\log \log n)^c)$ time.

- Approximate sparsity, $2^{-k}$ failure probability
  - Compressed sensing, using Restricted Isometry Property
  - $O(k \log^4 n)$ samples, $O(n \log^c n)$ time.
Classes of sparse Fourier transform algorithms
For recovering a $k$-sparse signal in $n$ dimensions.

- **Exact sparsity, deterministic algorithm**
  - Vandermonde matrix: $2k$ samples sufficient
  - Syndrome decoding for Reed-Solomon coding
  - Berlekamp-Massey: $O(k^2 + k(\log \log n)^c)$ time.

- **Approximate sparsity, $2^{-k}$ failure probability**
  - Compressed sensing, using Restricted Isometry Property
  - $O(k \log^4 n)$ samples, $O(n \log^c n)$ time.

**Today**: Approximate sparsity, $1/4$ or $1/n^c$ probability.
Classes of sparse Fourier transform algorithms

For recovering a $k$-sparse signal in $n$ dimensions.

- **Exact sparsity, deterministic algorithm**
  - Vandermonde matrix: $2k$ samples sufficient
  - Syndrome decoding for Reed-Solomon coding
  - Berlekamp-Massey: $O(k^2 + k(\log \log n)^c)$ time.

- **Approximate sparsity, $2^{-k}$ failure probability**
  - Compressed sensing, using Restricted Isometry Property
  - $O(k \log^4 n)$ samples, $O(n \log^c n)$ time.

**Today**: Approximate sparsity, $1/4$ or $1/n^c$ probability.
  - Using hashing
Classes of sparse Fourier transform algorithms
For recovering a $k$-sparse signal in $n$ dimensions.

- **Exact sparsity, deterministic algorithm**
  - Vandermonde matrix: $2k$ samples sufficient
  - Syndrome decoding for Reed-Solomon coding
  - Berlekamp-Massey: $O(k^2 + k(\log \log n)^c)$ time.

- **Approximate sparsity, $2^{-k}$ failure probability**
  - Compressed sensing, using Restricted Isometry Property
  - $O(k \log^4 n)$ samples, $O(n \log^c n)$ time.

**Today**: Approximate sparsity, $1/4$ or $1/n^c$ probability.
  - Using hashing
  - $O(k \log^c n)$ samples, $O(k \log^c n)$ time.
Kinds of Fourier transform

- 1d Fourier transform: \( x \in \mathbb{C}^n, \omega = e^{2\pi i/n} \), want

\[
\hat{x}_i = \sum_{j=1}^{n} \omega^{ij} x_j
\]
Kinds of Fourier transform

- 1d Fourier transform: $x \in \mathbb{C}^n$, $\omega = e^{2\pi i/n}$, want

\[
\hat{x}_i = \sum_{j=1}^{n} \omega^{ij} x_j
\]

- 2d Fourier Transform: $x \in \mathbb{C}^{n_1 \times n_2}$, $\omega_i = e^{2\pi i/n_i}$, want

\[
\hat{x}_{i_1,i_2} = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \omega_1^{i_1 j_1} \omega_2^{i_2 j_2} x_{j_1,j_2}
\]
Kinds of Fourier transform

1d Fourier transform: \( x \in \mathbb{C}^n, \omega = e^{2\pi i/n}, \) want

\[ \hat{x}_i = \sum_{j=1}^{n} \omega^{ij} x_j \]

2d Fourier Transform: \( x \in \mathbb{C}^{n_1 \times n_2}, \omega_i = e^{2\pi i/n_i}, \) want

\[ \hat{x}_{i_1,i_2} = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \omega_1^{i_1j_1} \omega_2^{i_2j_2} x_{j_1,j_2} \]
Kinds of Fourier transform

- **1d Fourier transform**: $x \in \mathbb{C}^n$, $\omega = e^{2\pi i/n}$, want
  \[
  \hat{x}_i = \sum_{j=1}^{n} \omega^{ij} x_j
  \]

- **2d Fourier Transform**: $x \in \mathbb{C}^{n_1 \times n_2}$, $\omega_i = e^{2\pi i/n_i}$, want
  \[
  \hat{x}_{i_1,i_2} = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \omega_1^{i_1j_1} \omega_2^{i_2j_2} x_{j_1,j_2}
  \]
Kinds of Fourier transform

- 1d Fourier transform: \( x \in \mathbb{C}^n, \omega = e^{2\pi i/n} \), want
  \[
  \hat{x}_i = \sum_{j=1}^{n} \omega^{ij} x_j
  \]

- 2d Fourier Transform: \( x \in \mathbb{C}^{n_1 \times n_2}, \omega_i = e^{2\pi i/n_i} \), want
  \[
  \hat{x}_{i_1,i_2} = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \omega_1^{i_1 j_1} \omega_2^{i_2 j_2} x_{j_1,j_2}
  \]

- If \( n_1, n_2 \) are relatively prime, equivalent to 1d transform of \( \mathbb{C}^{n_1 n_2} \)
Kinds of Fourier transform

- **1d Fourier transform**: \( x \in \mathbb{C}^n, \omega = e^{2\pi i/n}, \) want
  \[
  \hat{x}_i = \sum_{j=1}^{n} \omega^{ij} x_j
  \]

- **2d Fourier Transform**: \( x \in \mathbb{C}^{n_1 \times n_2}, \omega_i = e^{2\pi i/n_i}, \) want
  \[
  \hat{x}_{i_1,i_2} = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \omega_{1}^{i_1 j_1} \omega_{2}^{i_2 j_2} x_{j_1,j_2}
  \]
  - If \( n_1, n_2 \) are relatively prime, equivalent to 1d transform of \( \mathbb{C}^{n_1 n_2} \)

- **Hadamard transform**: \( x \in \mathbb{C}^{2 \times 2 \times \cdots \times 2} \)
  \[
  \hat{x}_i = \sum_{j}^{n} (-1)^{\langle i, j \rangle} x_j
  \]
Goal: given access to $x$, compute $\tilde{x} \approx \hat{x}$

- Exact case: $\hat{x}$ is $k$-sparse, $\tilde{x} = \hat{x}$ (maybe to log $n$ bits of precision)

- Approximate case: $\|x - \hat{x}\|_2 \leq (1 + \epsilon) \min_k \|\hat{x} - \hat{x}_k\|_2$

With "good" probability.

Algorithm for $k = 1$ (exact or approximate)

Method to reduce to $k = 1$ case

- Split $\hat{x}$ into $O(k)$ "random" parts
- Can sample time domain of the parts.
- $\mathcal{O}(k \log k)$ time to get one sample from each of the $k$ parts.

Finds "most" of signal; repeat on residual
Goal: given access to $x$, compute $\overline{x} \approx \hat{x}$

- Exact case: $\hat{x}$ is $k$-sparse, $\overline{x} = \hat{x}$ (maybe to log $n$ bits of precision)
- Approximate case:

$$\|\overline{x} - \hat{x}\|_2 \leq (1 + \epsilon) \min_{k\text{-sparse } \hat{x}_k} \|\hat{x} - \hat{x}_k\|_2$$
Goal: given access to $x$, compute $\bar{x} \approx \hat{x}$

- Exact case: $\hat{x}$ is $k$-sparse, $\bar{x} = \hat{x}$ (maybe to $\log n$ bits of precision)
- Approximate case:

$$\|\bar{x} - \hat{x}\|_2 \leq (1 + \epsilon) \min_{k\text{-sparse } \hat{x}_k} \|\hat{x} - \hat{x}_k\|_2$$

- With “good” probability.
Goal: given access to $x$, compute $\bar{x} \approx \hat{x}$

- Exact case: $\hat{x}$ is $k$-sparse, $\bar{x} = \hat{x}$ (maybe to log $n$ bits of precision)
- Approximate case:

$$\|\bar{x} - \hat{x}\|_2 \leq (1 + \epsilon) \min_{k\text{-sparse } \hat{x}_k} \|\hat{x} - \hat{x}_k\|_2$$

- With “good” probability.

1. Algorithm for $k = 1$ (exact or approximate)
Generic Algorithm Outline

- Goal: given access to $x$, compute $\bar{x} \approx \hat{x}$
  - Exact case: $\hat{x}$ is $k$-sparse, $\bar{x} = \hat{x}$ (maybe to log $n$ bits of precision)
  - Approximate case:
    
    $$\|\bar{x} - \hat{x}\|_2 \leq (1 + \epsilon) \min_{k\text{-sparse } \hat{x}_k} \|\hat{x} - \hat{x}_k\|_2$$

  - With “good” probability.

1. Algorithm for $k = 1$ (exact or approximate)
2. Method to reduce to $k = 1$ case
Generic Algorithm Outline

1. Algorithm for $k = 1$ (exact or approximate)
2. Method to reduce to $k = 1$ case
Generic Algorithm Outline

1. Algorithm for $k = 1$ (exact or approximate)
2. Method to reduce to $k = 1$ case
Generic Algorithm Outline

Algorithm for $k = 1$ (exact or approximate)

Method to reduce to $k = 1$ case
  - Split $\hat{x}$ into $O(k)$ “random” parts
Algorithm for $k = 1$ (exact or approximate)

1. Method to reduce to $k = 1$ case
   - Split $\hat{x}$ into $O(k)$ “random” parts
   - Can sample time domain of the parts.
Generic Algorithm Outline

1. Algorithm for \( k = 1 \) (exact or approximate)
2. Method to reduce to \( k = 1 \) case
   - Split \( \hat{x} \) into \( O(k) \) “random” parts
   - Can sample time domain of the parts.
     - \( O(k \log k) \) time to get one sample from each of the \( k \) parts.
Generic Algorithm Outline

1. Algorithm for $k = 1$ (exact or approximate)
2. Method to reduce to $k = 1$ case
   - Split $\hat{x}$ into $O(k)$ “random” parts
   - Can sample time domain of the parts.
     - $O(k \log k)$ time to get one sample from each of the $k$ parts.
3. Finds “most” of signal; repeat on residual
Talk Outline

1 Algorithm for $k = 1$
Talk Outline

1. Algorithm for $k = 1$

2. Reducing $k$ to 1
Talk Outline

1. Algorithm for $k = 1$
2. Reducing $k$ to 1
3. Putting it together
Talk Outline

1. Algorithm for $k = 1$
2. Reducing $k$ to 1
3. Putting it together
Algorithm for $k = 1$: one dimension, exact case

**Lemma**

*We can compute a 1-sparse $\hat{x}$ in $O(1)$ time.*

\[ \hat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \]
Algorithm for $k = 1$: one dimension, exact case

Lemma

*We can compute a 1-sparse $\hat{x}$ in $O(1)$ time.*

\[
\hat{x}_i = \begin{cases} 
  a & \text{if } i = t \\
  0 & \text{otherwise}
\end{cases}
\]

Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$. 

(Related to OFDM, Prony’s method, matrix pencil.)
Algorithm for $k = 1$: one dimension, exact case

Lemma

We can compute a 1-sparse $\hat{x}$ in $O(1)$ time.

$\hat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$

Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$.

$x_0 = a$
Algorithm for $k = 1$: one dimension, exact case

**Lemma**

*We can compute a 1-sparse $\hat{x}$ in $O(1)$ time.*

\[ \hat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \]

Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$.

- $x_0 = a$
- $x_1 = a\omega^t$
Algorithm for $k = 1$: one dimension, exact case

Lemma

We can compute a 1-sparse $\hat{x}$ in $O(1)$ time.

$$\hat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$.

$x_0 = a$ \hspace{1cm} $x_1 = a\omega^t$

$x_1/x_0 = \omega^t \implies t$. 

(Related to OFDM, Prony's method, matrix pencil.)
Lemma

*We can compute a 1-sparse* \( \hat{x} \) *in* \( O(1) \) *time.*

\[
\hat{x}_i = \begin{cases} 
  a & \text{if } i = t \\
  0 & \text{otherwise}
\end{cases}
\]

Then \( x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t}) \).

\[
x_0 = a \quad \quad x_1 = a\omega^t
\]

\( x_1/x_0 = \omega^t \implies t. \)

(Related to OFDM, Prony's method, matrix pencil.)
Algorithm for $k = 1$: one dimension, exact case

Lemma

We can compute a 1-sparse $\hat{x}$ in $O(1)$ time.

$$\hat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$.

$x_0 = a \quad x_1 = a\omega^t$

$x_1/x_0 = \omega^t \implies t$.

(Related to OFDM, Prony’s method, matrix pencil.)
Lemma

Suppose \( \hat{x} \) is approximately 1-sparse:

\[
\left| \hat{x}_t \right| / \| \hat{x} \|_2 \geq 90\%.
\]

Then we can recover it with \( O(\log n) \) samples and \( O(\log^2 n) \) time.

Algorithm for \( k = 1 \): one dimension, approximate case
Algorithm for $k = 1$: one dimension, approximate case

Lemma

Suppose $\hat{x}$ is approximately 1-sparse:

$$|\hat{x}_t|/\|\hat{x}\|_2 \geq 90\%.$$  

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.

- With exact sparsity: $\log n$ bits in a single measurement.
Algorithm for $k = 1$: one dimension, approximate case

Lemma

Suppose $\hat{x}$ is approximately 1-sparse:

$$|\hat{x}_t|/\|\hat{x}\|_2 \geq 90\%.$$  

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.

- With exact sparsity: $\log n$ bits in a single measurement.
- With noise: only constant number of useful bits.
Algorithm for $k = 1$: one dimension, approximate case

**Lemma**

Suppose $\hat{x}$ is approximately 1-sparse:

$$\frac{|\hat{x}_t|}{\|\hat{x}\|_2} \geq 90\%.$$ 

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.

- With exact sparsity: log $n$ bits in a single measurement.
- With noise: only constant number of useful bits.
- Choose $\Theta(\log n)$ time shifts $c$ to recover $i$. 
Algorithm for $k = 1$: one dimension, approximate case

**Lemma**

*Suppose* $\hat{x}$ *is approximately 1-sparse:*

$$\frac{|\hat{x}_t|}{\|\hat{x}\|_2} \geq 90\%.$$  

*Then we can recover it with* $O(\log n)$ *samples and* $O(\log^2 n)$ *time.*

- With exact sparsity: $\log n$ bits in a single measurement.
- With noise: only constant number of useful bits.
- Choose $\Theta(\log n)$ time shifts $c$ to recover $i$. 

$$x_{c_2}/x_0 = \omega^{c_2 t} + \text{noise}$$
Algorithm for $k = 1$: one dimension, approximate case

Lemma

Suppose $\hat{x}$ is approximately 1-sparse:

$$|\hat{x}_t|/\|\hat{x}\|_2 \geq 90\%.$$ 

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.

- With exact sparsity: log $n$ bits in a single measurement.
- With noise: only constant number of useful bits.
- Choose $\Theta(\log n)$ time shifts $c$ to recover $i$. 
Algorithm for $k = 1$: one dimension, approximate case

Lemma

Suppose $\hat{x}$ is approximately 1-sparse:

$$\frac{|\hat{x}_t|}{\|\hat{x}\|_2} \geq 90\%.$$  

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.

- With exact sparsity: log $n$ bits in a single measurement.
- With noise: only constant number of useful bits.
- Choose $\Theta(\log n)$ time shifts $c$ to recover $i$.
- Error correcting code with efficient recovery $\implies$ lemma.
Algorithm for $k = 1$: Hadamard setting

Levin ’93, improving upon Goldreich-Levin ’89

$$\hat{x}_i = \sum_j (-1)^{\langle i, j \rangle} x_j$$
Algorithm for $k = 1$: Hadamard setting

Levin ’93, improving upon Goldreich-Levin ’89

$$\hat{x}_i = \sum_j (-1)^{\langle i,j \rangle} x_j$$

Lemma

Suppose $\hat{x}$ is approximately 1-sparse:

$$|\hat{x}_t|/\|\hat{x}\|_2 \geq 90\%.$$  

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.
Algorithm for $k = 1$: Hadamard setting

Levin ’93, improving upon Goldreich-Levin ’89

$$\hat{x}_i = \sum_j (-1)^{\langle i, j \rangle} x_j$$

**Lemma**

Suppose $\hat{x}$ is approximately 1-sparse:

$$|\hat{x}_t| / \|\hat{x}\|_2 \geq 90\%.$$

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.

- We have $\text{sign}(\hat{x}_r) = \text{sign}((-1)^{\langle r, t \rangle} x_t)$ with 9/10 probability over $r$. 

Algorithm for $k = 1$: Hadamard setting

Levin ’93, improving upon Goldreich-Levin ’89

\[
\hat{x}_i = \sum_j (-1)^{\langle i, j \rangle} x_j
\]

Lemma

Suppose $\hat{x}$ is approximately 1-sparse:

\[
|\hat{x}_t|/\|\hat{x}\|_2 \geq 90%.
\]

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.

- We have $\text{sign}(\hat{x}_r) = \text{sign}((-1)^{\langle r, t \rangle} x_t)$ with 9/10 probability over $r$.
- Therefore for any $i$, with 8/10 probability over $r$,

\[
\text{sign}(\frac{\hat{x}_{i+r}}{\hat{x}_r}) = \text{sign}((-1)^{\langle i, t \rangle})
\]
Algorithm for $k = 1$: Hadamard setting

Levin ’93, improving upon Goldreich-Levin ’89

\[ \hat{x}_i = \sum_j (-1)^{\langle i, j \rangle} x_j \]

Lemma

Suppose $\hat{x}$ is approximately 1-sparse:

\[ |\hat{x}_t|/||\hat{x}||_2 \geq 90\% \]

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.

- We have \( \text{sign}(\hat{x}_r) = \text{sign}((-1)^{\langle r, t \rangle} x_t) \) with 9/10 probability over $r$.
- Therefore for any $i$, with 8/10 probability over $r$,
  \[ \text{sign}(\frac{\hat{x}_{i+r}}{\hat{x}_r}) = \text{sign}((-1)^{\langle i, t \rangle}) \]
- Choose $i$ to be the $O(\log n)$ rows of generator matrix for constant rate and distance binary code.
Talk Outline

1. Algorithm for $k = 1$

2. Reducing $k$ to 1

3. Putting it together
Algorithm for general $k$

- Reduce general $k$ to $k = 1$. 

![Diagram]

$x \xrightarrow{\text{Filters}} O(k) \xrightarrow{\text{1-sparse recovery}} \hat{x}'$

- "Filters": partition frequencies into $O(k)$ buckets.
- Sample from time domain of each bucket with $O(\log n)$ overhead.
- Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.
- Random permutation
- 1-sparse recovery
- 1-sparse recovery
- 1-sparse recovery
- 1-sparse recovery
- Recovers most of $\hat{x}$:

**Lemma (Partial sparse recovery)**

In $O(k \log n)$ expected time, we can compute an estimate $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.

![Diagram showing the process of reducing general $k$ to $k = 1$ through filters and 1-sparse recovery.]
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.

\[ x \xrightarrow{\text{Filters}} \begin{array}{c} 1\text{-sparse recovery} \\ 1\text{-sparse recovery} \\ \vdots \\ 1\text{-sparse recovery} \end{array} \xrightarrow{O(k)} \hat{x}' \]
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.

Diagram:
- Input $x$ to Filters
- $O(k)$ steps
- Output $\hat{x}'$
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm

$\hat{x}'$  

![Diagram showing the process of filtering and 1-sparse recovery]
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.

![Diagram of the algorithm with filters and 1-sparse recovery steps.]

**Lemma (Partial sparse recovery)**

In $O(k \log n)$ expected time, we can compute an estimate $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.

![Diagram of the algorithm](image)

**Filters**

1-sparse recovery

$O(k)$

1-sparse recovery

1-sparse recovery

1-sparse recovery

$x$ → Filters → $\hat{x}'$
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm

- Most frequencies alone in bucket.
- Random permutation

![Diagram of the algorithm](image)
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.
- Random permutation
Algorithm for general \( k \)

- Reduce general \( k \) to \( k = 1 \).
- “Filters”: partition frequencies into \( O(k) \) buckets.
  - Sample from time domain of each bucket with \( O(\log n) \) overhead.
  - Recovered by \( k = 1 \) algorithm
- Most frequencies alone in bucket.
- Random permutation

![Diagram of the algorithm](image)
Algorithm for general $k$

- Reduce general $k$ to $k = 1$.
- “Filters”: partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm

Most frequencies alone in bucket.

- Random permutation

Recovers most of $\hat{x}$:

Lemma (Partial sparse recovery)

In $O(k \log n)$ expected time, we can compute an estimate $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.
Going from finding most coordinates to finding all

**Partial $k$-sparse recovery**

$x \xrightarrow{\text{Permute}} \text{Filters} \xrightarrow{O(k)} \hat{x}'$

**Lemma (Partial sparse recovery)**

\[ \text{In } O(k \log n) \text{ expected time, we can compute an estimate } \hat{x}' \text{ such that } \hat{x} - \hat{x}' \text{ is } k/2\text{-sparse.} \]
Lemma (Partial sparse recovery)

In $O(k \log n)$ expected time, we can compute an estimate $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.
Going from finding most coordinates to finding all \( \hat{x} - \hat{x}' \)

**Partial \( k \)-sparse recovery**

1-sparse recovery

\( O(k) \)

1-sparse recovery

1-sparse recovery

\( \hat{x}' \)

**Lemma (Partial sparse recovery)**

*In \( O(k \log n) \) expected time, we can compute an estimate \( \hat{x}' \) such that \( \hat{x} - \hat{x}' \) is \( k/2 \)-sparse.*

Repeat, \( k \to k/2 \to k/4 \to \ldots \)
Going from finding most coordinates to finding all.

**Partial $k$-sparse recovery**

$x \xrightarrow{\text{Permute}} \text{Filters} \xrightarrow{O(k)} \hat{x}'$

**Lemma (Partial sparse recovery)**

*In $O(k \log n)$ expected time, we can compute an estimate $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.*

Repeat, $k \rightarrow k/2 \rightarrow k/4 \rightarrow \cdots$
Going from finding most coordinates to finding all

Partial $k$-sparse recovery

Lemma (Partial sparse recovery)

In $O(k \log n)$ expected time, we can compute an estimate $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.

Repeat, $k \rightarrow k/2 \rightarrow k/4 \rightarrow \cdots$
Going from finding most coordinates to finding all

**Partial $k$-sparse recovery**

\[
\begin{align*}
&x \\ &\xrightarrow{\text{Permute}} \\
&\xrightarrow{\text{Filters}} \\
&\xrightarrow{O(k)} \\
&\quad \xrightarrow{\text{1-sparse recovery}} \\
&\quad \xrightarrow{\text{1-sparse recovery}} \\
&\quad \xrightarrow{\text{1-sparse recovery}} \\
&\quad \xrightarrow{\text{1-sparse recovery}} \\
&\xrightarrow{\hat{x}'}
\end{align*}
\]

**Lemma (Partial sparse recovery)**

In $O(k \log n)$ expected time, we can compute an estimate $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.

Repeat, $k \rightarrow k/2 \rightarrow k/4 \rightarrow \cdots$
Going from finding most coordinates to finding all

**Partial $k$-sparse recovery**

Lemma (Partial sparse recovery)

In $O(k \log n)$ expected time, we can compute an estimate $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.

Repeat, $k \rightarrow k/2 \rightarrow k/4 \rightarrow \cdots$

**Theorem**

We can compute $\hat{x}$ in $O(k \log n)$ expected time.
Going from finding most coordinates to finding all \( \hat{x} - \hat{x}' \)

Partial \( k \)-sparse recovery

\[ x \xrightarrow{\text{Permute}} \text{Filters} \xrightarrow{O(k)} \hat{x}' \]

\[ \hat{x} - \hat{x}' \text{ is } k/2\text{-sparse.} \]

Repeat, \( k \rightarrow k/2 \rightarrow k/4 \rightarrow \cdots \)

**Lemma (Partial sparse recovery)**

*In \( O(k \log n) \) expected time, we can compute an estimate \( \hat{x}' \) such that \( \hat{x} - \hat{x}' \) is \( k/2 \)-sparse.*

**Theorem**

*We can compute \( \hat{x} \) in \( O(k \log n) \) expected time.*
How do filters work?

Consider the $\sqrt{n} \times \sqrt{n}$ 2d setting.
How do filters work?

- Consider the $\sqrt{n} \times \sqrt{n}$ 2d setting.
- Get answer by FFT on rows, then FFT on resulting columns.
How do filters work?

- Consider the $\sqrt{n} \times \sqrt{n}$ 2d setting.
- Get answer by FFT on rows, then FFT on resulting columns.
How do filters work?

- Consider the $\sqrt{n} \times \sqrt{n}$ 2d setting.
- Get answer by FFT on rows, then FFT on resulting columns.
- What if I just take the FFT $y^r$ of a random row $r$?
How do filters work?

- Consider the $\sqrt{n} \times \sqrt{n}$ 2d setting.
- Get answer by FFT on rows, then FFT on resulting columns.
- What if I just take the FFT $y^r$ of a random row $r$?
- For any column $\hat{z} = \hat{x}_{*,c} \in \mathbb{C}^{\sqrt{n}}$ we have in the corresponding time domain

$$z_r = y^r_c$$
How do filters work?

- Consider the $\sqrt{n} \times \sqrt{n}$ 2d setting.
- Get answer by FFT on rows, then FFT on resulting columns.
- What if I just take the FFT $y^r$ of a random row $r$?
- For any column $\hat{z} = \hat{x}_{*,c} \in \mathbb{C}^{\sqrt{n}}$ we have in the corresponding time domain
  
  $$z_r = y^r_c$$

- With $O(\sqrt{n} \log n)$ time, get samples from time domains of all $\sqrt{n}$ columns.
How do filters work?

- Consider the $\sqrt{n} \times \sqrt{n}$ 2d setting.
- Get answer by FFT on rows, then FFT on resulting columns.
- What if I just take the FFT $y^r$ of a random row $r$?
- For any column $\hat{z} = \hat{x}_{*,c} \in \mathbb{C}^{\sqrt{n}}$ we have in the corresponding time domain
  \[ z_r = y_c^r \]
- With $O(\sqrt{n} \log n)$ time, get samples from time domains of all $\sqrt{n}$ columns.
- If column is 1-sparse, recover it with $O(1)$ row FFTs.
How do filters work?

Consider the $\sqrt{n} \times \sqrt{n}$ 2d setting.

Get answer by FFT on rows, then FFT on resulting columns.

What if I just take the FFT $y^r$ of a random row $r$?

For any column $\hat{z} = \hat{x}_{*,c} \in \mathbb{C}^{\sqrt{n}}$ we have in the corresponding time domain

$$z_r = y_c^r$$

With $O(\sqrt{n} \log n)$ time, get samples from time domains of all $\sqrt{n}$ columns.

If column is 1-sparse, recover it with $O(1)$ row FFTs

- For approximate sparsity, $O(\log n)$ row FFTs.
How do filters work?

- Consider the $\sqrt{n} \times \sqrt{n}$ 2d setting.
- Get answer by FFT on rows, then FFT on resulting columns.
- What if I just take the FFT $y^r$ of a random row $r$?
- For any column $\hat{z} = \hat{x}_{*,c} \in \mathbb{C}^{\sqrt{n}}$ we have in the corresponding time domain
  
  $$z_r = y^r_c$$

- With $O(\sqrt{n} \log n)$ time, get samples from time domains of all $\sqrt{n}$ columns.
- If column is 1-sparse, recover it with $O(1)$ row FFTs
  - For approximate sparsity, $O(\log n)$ row FFTs.
- If $k = \sqrt{n}$ random nonzeros, expect to recover most of them.
Filters more generally

- Fourier transform switches multiplication and convolution.

Choose a filter \(F\) so both \(F\) and \(\hat{F}\) are sparse

- \(F\) is \(\tilde{O}(k)\)-sparse and \(\hat{F}\) is (approximately) \(O(n/k)\)-sparse.

Last slide: \(F\) is row, \(\hat{F}\) is column

For various \(r\), compute \(k\)-dimensional Fourier transform of 

\[ y_i = x_i + rF_i. \]

Gives the \(r\)th time domain sample of \(\hat{x}\) · shift \((\hat{F})\) for \(k\) shifts of \(\hat{F}\).
Filters more generally

- Fourier transform switches multiplication and convolution.
- Choose a filter $F$ so both $F$ and $\hat{F}$ are sparse
Filters more generally

- Fourier transform switches multiplication and convolution.
- Choose a filter $F$ so both $F$ and $\hat{F}$ are sparse
  - $F$ is $\tilde{O}(k)$-sparse and $\hat{F}$ is (approximately) $O(n/k)$ sparse.
Filters more generally

- Fourier transform switches multiplication and convolution.
- Choose a filter $F$ so both $F$ and $\hat{F}$ are sparse
  - $F$ is $\tilde{O}(k)$-sparse and $\hat{F}$ is (approximately) $O(n/k)$ sparse.
  - Last slide: $F$ is row, $\hat{F}$ is column
Filters more generally

- Fourier transform switches multiplication and convolution.
- Choose a filter $F$ so both $F$ and $\hat{F}$ are sparse
  - $F$ is $\tilde{O}(k)$-sparse and $\hat{F}$ is (approximately) $O(n/k)$ sparse.
  - Last slide: $F$ is row, $\hat{F}$ is column

| $y_i = x_i + r F_i$ | Gives the $r$th time domain sample of $\hat{x}$·shift($\hat{F}$) for $k$ shifts of $\hat{F}$ |
Filters more generally

- Fourier transform switches multiplication and convolution.
- Choose a filter $F$ so both $F$ and $\hat{F}$ are sparse
  - $F$ is $\tilde{O}(k)$-sparse and $\hat{F}$ is (approximately) $O(n/k)$ sparse.
  - Last slide: $F$ is row, $\hat{F}$ is column
- For various $r$, compute $k$-dimensional Fourier transform of $y_i = x_{i+r}F_i$. 
Filters more generally

- Fourier transform switches multiplication and convolution.
- Choose a filter $F$ so both $F$ and $\hat{F}$ are sparse
  - $F$ is $\tilde{O}(k)$-sparse and $\hat{F}$ is (approximately) $O(n/k)$ sparse.
  - Last slide: $F$ is row, $\hat{F}$ is column
- For various $r$, compute $k$-dimensional Fourier transform of $y_i = x_{i+r}F_i$.
- Gives the $r$th time domain sample of $\hat{x} \cdot \text{shift}(\hat{F})$ for $k$ shifts of $\hat{F}$. 

Hadamard setting: full algorithm

- \( F = \text{span}(A) \) for any \( A \in \mathbb{F}_2^{\log n \times \log B} \)

For any \( r \in \text{span}(A) \), compute Hadamard transform of \( y_i = x_{Ai} + r \).

Gives \( r \)th time domain sample of \( \hat{x} \) restricted to all \( B \) cosets of \( A \).

If \( A \) is chosen randomly, then any two \( i, j \) land in same coset with probability \( 1/B \).

Each coordinate is alone with probability \( 1 - k/B \).

Take \( \log(n/k) \) different \( r \) to solve the 1-sparse problem on coset.

For \( B = O(k) \), expect to recover "most" coordinates.

Takes \( O(k \log(n/k)) \) samples and \( O(k \log(n/k) \log k) \) time.
Hadamard setting: full algorithm

- $F = \text{span}(A)$ for any $A \in \mathbb{F}_2^{\log n \times \log B}$
- For any $r \in \text{span}(A)^\perp$, compute Hadamard transform of
  
  $$y_i = x_{Ai+r}$$
Hadamard setting: full algorithm

- $F = \text{span}(A)$ for any $A \in \mathbb{F}_2^{\log n \times \log B}$
- For any $r \in \text{span}(A)^\perp$, compute Hadamard transform of
  $$y_i = x_{ Ai + r}$$
- Gives $r$th time domain sample of $\hat{x}$ restricted to all $B$ cosets of $A^\perp$. 

Eric Price  
Tutorial on Sparse Fourier Transforms  
19 / 27
Hadamard setting: full algorithm

- $F = \text{span}(A)$ for any $A \in \mathbb{F}_2^{\log n \times \log B}$
- For any $r \in \text{span}(A)^\perp$, compute Hadamard transform of

$$y_i = x_{A_i + r}$$

- Gives $r$th time domain sample of $\hat{x}$ restricted to all $B$ cosets of $A^\perp$.
- If $A$ is chosen randomly, then any two $i, j$ land in same coset with probability $1/B$. 
Hadamard setting: full algorithm

- \( F = \text{span}(A) \) for any \( A \in \mathbb{F}_2^{\log n \times \log B} \)
- For any \( r \in \text{span}(A)^\perp \), compute Hadamard transform of

\[
y_i = x_{Ai} + r
\]

- Gives \( r \)th time domain sample of \( \hat{x} \) restricted to all \( B \) cosets of \( A^\perp \).
- If \( A \) is chosen randomly, then any two \( i, j \) land in same coset with probability \( 1/B \).
- Each coordinate is alone with probability \( 1 - k/B \).
Hadamard setting: full algorithm

- $F = \text{span}(A)$ for any $A \in \mathbb{F}_2^{\log n \times \log B}$
- For any $r \in \text{span}(A)^\perp$, compute Hadamard transform of

$$y_i = x_{Ai} + r$$

- Gives $r$th time domain sample of $\hat{x}$ restricted to all $B$ cosets of $A^\perp$.
- If $A$ is chosen randomly, then any two $i, j$ land in same coset with probability $1/B$.
- Each coordinate is alone with probability $1 - k/B$.
- Take $\log(n/k)$ different $r$ to solve the 1-sparse problem on coset.
Hadamard setting: full algorithm

- $F = \text{span}(A)$ for any $A \in \mathbb{F}_2^{\log n \times \log B}$
- For any $r \in \text{span}(A)^\perp$, compute Hadamard transform of

$$y_i = x_{Ai + r}$$

- Gives $r$th time domain sample of $\hat{x}$ restricted to all $B$ cosets of $A^\perp$.
- If $A$ is chosen randomly, then any two $i, j$ land in same coset with probability $1/B$.
- Each coordinate is alone with probability $1 - k/B$.
- Take $\log(n/k)$ different $r$ to solve the 1-sparse problem on coset.
- For $B = O(k)$, expect to recover “most” coordinates.
Hadamard setting: full algorithm

- $F = \text{span}(A)$ for any $A \in \mathbb{F}_2^{\log n \times \log B}$
- For any $r \in \text{span}(A)^\perp$, compute Hadamard transform of
  
  $$y_i = x_{Ai+r}$$

- Gives $r$th time domain sample of $\hat{x}$ restricted to all $B$ cosets of $A^\perp$.
- If $A$ is chosen randomly, then any two $i, j$ land in same coset with probability $1/B$.
- Each coordinate is alone with probability $1 - k/B$.
- Take $\log(n/k)$ different $r$ to solve the 1-sparse problem on coset.
- For $B = O(k)$, expect to recover “most” coordinates.
- Takes $O(k \log(n/k))$ samples and $O(k \log(n/k) \log k)$ time
Hadamard setting: full algorithm

- $F = \text{span}(A)$ for any $A \in \mathbb{F}_2^{\log n \times \log B}$
- For any $r \in \text{span}(A)\perp$, compute Hadamard transform of

$$y_i = x_{Ai+r}$$

- Gives $r$th time domain sample of $\hat{x}$ restricted to all $B$ cosets of $A\perp$.
- If $A$ is chosen randomly, then any two $i, j$ land in same coset with probability $1/B$.
- Each coordinate is alone with probability $1 - k/B$.
- Take $\log(n/k)$ different $r$ to solve the 1-sparse problem on coset.
- For $B = O(k)$, expect to recover “most” coordinates.
- Takes $O(k \log(n/k))$ samples and $O(k \log(n/k) \log k)$ time
- Repeat with $k \rightarrow k/2 \rightarrow k/4 \rightarrow \ldots$
Hadamard setting: full algorithm

- $F = \text{span}(A)$ for any $A \in \mathbb{F}_2^{\log n \times \log B}$
- For any $r \in \text{span}(A)^\perp$, compute Hadamard transform of $y_i = x_{Ai+r}$

Gives $r$th time domain sample of $\hat{x}$ restricted to all $B$ cosets of $A^\perp$.

If $A$ is chosen randomly, then any two $i, j$ land in same coset with probability $1/B$.

Each coordinate is alone with probability $1 - k/B$.

Take $\log(n/k)$ different $r$ to solve the 1-sparse problem on coset.

For $B = O(k)$, expect to recover “most” coordinates.

Takes $O(k \log(n/k))$ samples and $O(k \log(n/k) \log k)$ time

Repeat with $k \rightarrow k/2 \rightarrow k/4 \rightarrow \ldots$

Gives $O(k \log(n/k))$ total samples and $O(k \log(n/k) \log k)$ time
Hadamard setting: full algorithm

- \( F = \text{span}(A) \) for any \( A \in \mathbb{F}_2^{\log n \times \log B} \)
- For any \( r \in \text{span}(A)^\perp \), compute Hadamard transform of
  \[
y_i = x_{Ai} + r
  \]
  Gives \( r \)th time domain sample of \( \hat{x} \) restricted to all \( B \) cosets of \( A^\perp \).
  - If \( A \) is chosen randomly, then any two \( i, j \) land in same coset with probability 1/\( B \).
  - Each coordinate is alone with probability 1 − \( k/B \).
  - Take \( \log(n/k) \) different \( r \) to solve the 1-sparse problem on coset.
  - For \( B = O(k) \), expect to recover “most” coordinates.
  - Takes \( O(k \log(n/k)) \) samples and \( O(k \log(n/k) \log k) \) time
  - Repeat with \( k \to k/2 \to k/4 \to \ldots \)
  - Gives \( O(k \log(n/k)) \) total samples and \( O(k \log(n/k) \log k) \) time
Difficulty in other settings

- Not enough filters $F$ that are “perfect” ($F$ and $\hat{F}$ are indicators)
Difficulty in other settings

- Not enough filters $F$ that are “perfect” ($F$ and $\hat{F}$ are indicators)
- Two dimensions:
Difficulty in other settings

- Not enough filters $F$ that are “perfect” ($F$ and $\hat{F}$ are indicators)
- Two dimensions:
  - Can look at the columns or the rows
Difficulty in other settings

- Not enough filters $F$ that are “perfect” ($F$ and $\hat{F}$ are indicators)
- Two dimensions:
  - Can look at the columns or the rows
  - Some inputs will cause collisions for any projection.
Difficulty in other settings

- Not enough filters $F$ that are “perfect” ($F$ and $\hat{F}$ are indicators)
- Two dimensions:
  - Can look at the columns or the rows
  - Some inputs will cause collisions for any projection.
- Works if you assume coordinates randomly distributed [GHIKPS, Pawar-Ramchandran]
Difficulty in other settings

- Not enough filters $F$ that are “perfect” ($F$ and $\hat{F}$ are indicators)
- Two dimensions:
  - Can look at the columns or the rows
  - Some inputs will cause collisions for any projection.
- Works if you assume coordinates randomly distributed [GHIKPS, Pawar-Ramchandran]
  - Peeling procedure
Difficulty in other settings

- Not enough filters $F$ that are “perfect” ($F$ and $\hat{F}$ are indicators)
- Two dimensions:
  - Can look at the columns or the rows
  - Some inputs will cause collisions for any projection.
- Works if you assume coordinates randomly distributed [GHIKPS, Pawar-Ramchandran]
  - Peeling procedure
  - Still doesn’t work for 1 dimension, $n = 2^\ell$. 

For worst-case inputs, need other filters
Difficulty in other settings

- Not enough filters $F$ that are “perfect” ($F$ and $\hat{F}$ are indicators)
- Two dimensions:
  - Can look at the columns or the rows
  - Some inputs will cause collisions for any projection.
- Works if you assume coordinates randomly distributed [GHIKPS, Pawar-Ramchandran]
  - Peeling procedure
  - Still doesn’t work for 1 dimension, $n = 2^\ell$.
- For worst-case inputs, need other filters
A different style of filter
GMS05, HIKP12, IKP14, IK14

Filter (time): \( k \) uniformly spaced

Filter (frequency): \( n/k \) uniformly spaced

Previous slides used \textit{comb filter}
A different style of filter
GMS05, HIKP12, IKP14, IK14

- Filter (time): Gaussian \cdot \text{sinc}
- Filter (frequency): Gaussian \ast \text{rectangle}

- Previous slides used *comb filter*
- Instead, make filter so \( \hat{F} \) is large on an *interval.*
A different style of filter
GMS05, HIKP12, IKP14, IK14

Previous slides used *comb filter*
Instead, make filter so $\hat{F}$ is large on an *interval*.
We can permute the frequencies:

$$x'_i = x_{\sigma i} \implies \hat{x}_i = \hat{x}_{\sigma^{-1} i}$$
A different style of filter

GMS05, HIKP12, IKP14, IK14

- Previous slides used *comb filter*
- Instead, make filter so $\hat{F}$ is large on an *interval*.
- We can permute the frequencies:

  $$x_i' = x_{\sigma i} \implies \hat{x}_i = \hat{x}_{\sigma^{-1} i}$$

- This changes the coordinates in an interval (unlike in a comb).
A different style of filter
GMS05, HIKP12, IKP14, IK14

- Previous slides used \textit{comb filter}
- Instead, make filter so $\hat{F}$ is large on an \textit{interval}.
- We can permute the frequencies:

$$x'_i = x_{\sigma i} \implies \hat{x}_i = \hat{x}_{\sigma^{-1} i}$$

- This changes the coordinates in an interval (unlike in a comb).
- Allows us to convert worst case to random case.
1 Algorithm for $k = 1$

2 Reducing $k$ to 1

3 Putting it together
How can you hope for sublinear time?

$n$-dimensional DFT: $O(n \log n)$

$x \rightarrow \hat{x}$
How can you hope for sublinear time?

$n$-dimensional DFT: $O(n \log n)$

$x \rightarrow \hat{x}$
How can you hope for sublinear time?

$n$-dimensional DFT: $O(n \log n)$
$x \rightarrow \hat{x}$

$n$-dimensional DFT of first $k$ terms: $O(n \log n)$
$x \cdot \text{rect} \rightarrow \hat{x} \ast \text{sinc.}$
How can you hope for sublinear time?

**n-dimensional DFT:**

$O(n \log n)$

$x \rightarrow \hat{x}$

**n-dimensional DFT of first $k$ terms:**

$O(B \log B)$

$\ x \cdot \text{rect} \rightarrow \hat{x} \ast \text{sinc.}$
How can you hope for sublinear time?

\[ \text{n-dimensional DFT: } O(n \log n) \]
\[ x \rightarrow \hat{x} \]

\[ \text{n-dimensional DFT of first } k \text{ terms: } O(n \log n) \]
\[ x \cdot \text{rect} \rightarrow \hat{x} \ast \text{sinc.} \]

\[ \text{k-dimensional DFT of first } k \text{ terms: } O(B \log B) \]
\[ \text{alias}(x \cdot \text{rect}) \rightarrow \text{subsample}(\hat{x} \ast \text{sinc}). \]
How can you hope for sublinear time?

$n$-dimensional DFT: $O(n \log n)$
$x \rightarrow \hat{x}$

$n$-dimensional DFT of first $k$ terms: $O(n \log n)$
$x \cdot \text{rect} \rightarrow \hat{x} \ast \text{sinc.}$

$k$-dimensional DFT of first $k$ terms: $O(B \log B)$
alias$(x \cdot \text{rect}) \rightarrow$ subsample$(\hat{x} \ast \text{sinc})$. 
Algorithm for exactly sparse signals

Original signal $x$

Goal $\hat{x}$

Lemma
If $t$ is isolated in its bucket and in the "super-pass" region, the value $b$ we compute for its bucket satisfies $b = \hat{x}_t$.

Computing the $b$ for all $O(k \log n)$ buckets takes $O(k \log n)$ time.
Algorithm for exactly sparse signals

Computed $F \cdot x$

Filtered signal $\hat{F} \ast \hat{x}$

Lemma
If $t$ is isolated in its bucket and in the "super-pass" region, the value $b$ we compute for its bucket satisfies $b = \hat{x}_t$.

Computing the $b$ for all $O(k)$ buckets takes $O(k \log n)$ time.
Algorithm for exactly sparse signals

\[ F \cdot x \text{ aliased to } k \text{ terms} \]

Filtered signal \( \hat{F} \ast \hat{x} \)

Lemma

If \( t \) is isolated in its bucket and in the "super-pass" region, the value \( b \) we compute for its bucket satisfies

\[ b = \hat{x}_t. \]

Computing the \( b \) for all \( O(k) \) buckets takes \( O(k \log n) \) time.
Algorithm for exactly sparse signals

$F \cdot x$ aliased to $k$ terms

Computed samples of $\hat{F} \ast \hat{x}$

Lemma
If $t$ is isolated in its bucket and in the "super-pass" region, the value $b$ we compute for its bucket satisfies

$$b = \hat{x}_t.$$ 

Computing the $b$ for all $O(k)$ buckets takes $O(k \log n)$ time.
Algorithm for exactly sparse signals

$F \cdot x$ aliased to $k$ terms

Computed samples of $\hat{F} \ast \hat{x}$

Lemma

If $t$ is isolated in its bucket and in the "super-pass" region, the value $b$ we compute for its bucket satisfies

$$b = \hat{x}_t.$$  

Computing the $b$ for all $O(k)$ buckets takes $O(k \log n)$ time.
Algorithm for exactly sparse signals

\[ F \cdot x \text{ aliased to } k \text{ terms} \]

Knowledge about \( \hat{x} \)
Algorithm for *exactly sparse* signals

\[ F \cdot x \] aliased to \( k \) terms

Knowledge about \( \hat{x} \)

Lemma

If \( t \) is isolated in its bucket and in the "super-pass" region, the value \( b \) we compute for its bucket satisfies

\[ b = \hat{x}_t. \]

Computing the \( b \) for all \( O(k) \) buckets takes \( O(k \log n) \) time.
Algorithm for exactly sparse signals

$F \cdot x$ aliased to $k$ terms

Knowledge about $\hat{x}$

Lemma

*If* $t$ *is isolated in its bucket and in the “super-pass” region, the value* $b$ *we compute for its bucket satisfies*

\[ b = \hat{x}_t. \]

*Computing the* $b$ *for all* $O(k)$ *buckets takes* $O(k \log n)$ *time.*
Algorithm

Lemma

For most $t$, the value $b$ we compute for its bucket satisfies

$$b = \hat{x}_t.\]

Computing the $b$ for all $O(k)$ buckets takes $O(k \log n)$ time.
Algorithm

Lemma

For most $t$, the value $b$ we compute for its bucket satisfies

$$b = \hat{x}_t.$$  

Computing the $b$ for all $O(k)$ buckets takes $O(k \log n)$ time.

- Time-shift $x$ by one and repeat: $b' = \hat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t$.


**Algorithm**

**Lemma**

*For most* $t$, the value $b$ we compute for its bucket satisfies*

$$ b = \hat{x}_t. $$

*Computing the $b$ for all $O(k)$ buckets takes $O(k \log n)$ time.*

- Time-shift $x$ by one and repeat: $b' = \hat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t \implies$ can compute $t$. 
Lemma

For most $t$, the value $b$ we compute for its bucket satisfies

$$b = \hat{x}_t.$$  

Computing the $b$ for all $O(k)$ buckets takes $O(k \log n)$ time.

- Time-shift $x$ by one and repeat: $b' = \hat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t \implies$ can compute $t$.
  - Just like our 1-sparse recovery algorithm, $x_1/x_0 = \omega^t$.  

Eric Price
Tutorial on Sparse Fourier Transforms
25 / 27
Algorithm

Lemma

For most $t$, the value $b$ we compute for its bucket satisfies

$$b = \hat{x}_t.$$ 

Computing the $b$ for all $O(k)$ buckets takes $O(k \log n)$ time.

- Time-shift $x$ by one and repeat: $b' = \hat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t \implies$ can compute $t$.
  - Just like our 1-sparse recovery algorithm, $x_1/x_0 = \omega^t$.
- Gives partial sparse recovery: $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.
Algorithm

Lemma

For most $t$, the value $b$ we compute for its bucket satisfies

$$ b = \hat{x}_t. $$

Computing the $b$ for all $O(k)$ buckets takes $O(k \log n)$ time.

- Time-shift $x$ by one and repeat: $b' = \hat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t \implies$ can compute $t$.
  - Just like our 1-sparse recovery algorithm, $x_1/x_0 = \omega^t$.
- Gives partial sparse recovery: $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.

Repeat $k \rightarrow k/2 \rightarrow k/4 \rightarrow \cdots$
Algorithm

Lemma

For most $t$, the value $b$ we compute for its bucket satisfies

$$b = \hat{x}_t.$$ 

Computing the $b$ for all $O(k)$ buckets takes $O(k \log n)$ time.

- Time-shift $x$ by one and repeat: $b' = \hat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t \implies$ can compute $t$.
  - Just like our 1-sparse recovery algorithm, $x_1/x_0 = \omega^t$.
- Gives partial sparse recovery: $\hat{x}'$ such that $\hat{x} - \hat{x}'$ is $k/2$-sparse.

Repeat $k \rightarrow k/2 \rightarrow k/4 \rightarrow \cdots$

$O(k \log n)$ time sparse Fourier transform.

Eric Price
Tutorial on Sparse Fourier Transforms
State of the Art

- Algorithms based on two kinds of filters:
  
  ▶ Comb filter works for Hadamard transform in the worst case and \( n = (pq)\ell \), in the average case.
  
  ▶ Interval filter works for constant dimensional transform in the worst case, where \( n \) has \( \Theta(k) \)-sized factors.

Exactly sparse: “optimal” is \( O(k) \) samples and \( O(k \log k) \) time (and \( \log (n/k) \) factor larger for Hadamard).

▶ Comb filter: optimal when it works

▶ Interval filter: \( O(k \log n) \) samples and time

Approximately sparse: “optimal” is \( O(k \log (n/k)) \) samples and \( O(k \log (n/k) \log n) \) time.
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
  - Interval filter works for

\[ n = \left( \frac{pq}{\ell} \right) \]

Exactly sparse: "optimal" is \( O(k) \) samples and \( O(k \log k) \) time

Approximately sparse: "optimal" is \( O(k \log \left( \frac{n}{k} \right)) \) samples and \( O(k \log \left( \frac{n}{k} \right) \log n) \) time

Comb filter: optimal when it works

Interval filter: optimal samples OR optimal time OR \( \log c \log n \)-competitive mixture.
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
    - $>1$ dimensional transform, or $n = (pq)^\ell$, in the average case.
  - Interval filter works for
    - Constant dimensional transform in the worst case, $n$ has $\Theta(k)$-sized factors.

- Exactly sparse: "optimal" is $O(k)$ samples and $O(k \log k)$ time (and $\log(n/k)$ factor larger for Hadamard).

- Approximately sparse: "optimal" is $O(k \log(n/k))$ samples and $O(k \log(n/k) \log n)$ time.

- Comb filter: optimal when it works

- Interval filter: $O(k \log n)$ samples and $O(k \log(n/k) \log n)$ time (and $\log c \log n$-competitive mixture).
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
    - > 1 dimensional transform, or \( n = (pq)^\ell \), in the average case.
  - Interval filter works for
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
    - \( n > 1 \) dimensional transform, or \( n = (pq)^\ell \), in the average case.
  - Interval filter works for
    - Constant dimensional transform in the worst case, \( n \) has \( \Theta(k) \)-sized factors.
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
    - $> 1$ dimensional transform, or $n = (pq)^\ell$, in the average case.
  - Interval filter works for
    - Constant dimensional transform in the worst case, $n$ has $\Theta(k)$-sized factors.

- Exactly sparse: “optimal” is $O(k)$ samples and $O(k \log k)$ time (and $\log(n/k)$ factor larger for Hadamard)
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
    - $> 1$ dimensional transform, or $n = (pq)^\ell$, in the average case.
  - Interval filter works for
    - Constant dimensional transform in the worst case, $n$ has $\Theta(k)$-sized factors.

- Exactly sparse: “optimal” is $O(k)$ samples and $O(k \log k)$ time (and $\log(n/k)$ factor larger for Hadamard)
  - Comb filter: optimal when it works
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
    - $> 1$ dimensional transform, or $n = (pq)^\ell$, in the average case.
  - Interval filter works for
    - Constant dimensional transform in the worst case, $n$ has $\Theta(k)$-sized factors.

- Exactly sparse: “optimal” is $O(k)$ samples and $O(k \log k)$ time (and $\log(n/k)$ factor larger for Hadamard)
  - Comb filter: optimal when it works
  - Interval filter: $O(k \log n)$ samples and time
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
    - \( \ell \) dimensional transform, or \( n = (pq)^\ell \), in the average case.
  - Interval filter works for
    - Constant dimensional transform in the worst case, \( n \) has \( \Theta(k) \)-sized factors.

Exactly sparse: “optimal” is \( O(k) \) samples and \( O(k \log k) \) time (and \( \log(n/k) \) factor larger for Hadamard)
  - Comb filter: optimal when it works
  - Interval filter: \( O(k \log n) \) samples and time

Approximately sparse: “optimal” is \( O(k \log(n/k)) \) samples and \( O(k \log(n/k) \log n) \) time
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
    - \( n = (pq)^\ell \), in the average case.
  - Interval filter works for
    - Constant dimensional transform in the worst case, \( n \) has \( \Theta(k) \)-sized factors.

- Exactly sparse: “optimal” is \( O(k) \) samples and \( O(k \log k) \) time (and \( \log(n/k) \) factor larger for Hadamard)
  - Comb filter: optimal when it works
  - Interval filter: \( O(k \log n) \) samples and time

- Approximately sparse: “optimal” is \( O(k \log(n/k)) \) samples and \( O(k \log(n/k) \log n) \) time
  - Comb filter: optimal when it works
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
    - > 1 dimensional transform, or \( n = (pq)^\ell \), in the average case.
  - Interval filter works for
    - Constant dimensional transform in the worst case, \( n \) has \( \Theta(k) \)-sized factors.

Exactly sparse: “optimal” is \( O(k) \) samples and \( O(k \log k) \) time (and \( \log(n/k) \) factor larger for Hadamard)
  - Comb filter: optimal when it works
  - Interval filter: \( O(k \log n) \) samples and time

Approximately sparse: “optimal” is \( O(k \log(n/k)) \) samples and \( O(k \log(n/k) \log n) \) time
  - Comb filter: optimal when it works
  - Interval filter: optimal samples OR optimal time OR \( \log^c \log n \)-competitive mixture.
State of the Art

- Algorithms based on two kinds of filters:
  - Comb filter works for
    - Hadamard transform in the worst case
    - $> 1$ dimensional transform, or $n = (pq)^\ell$, in the average case.
  - Interval filter works for
    - Constant dimensional transform in the worst case, $n$ has $\Theta(k)$-sized factors.

Exactly sparse: “optimal” is $O(k)$ samples and $O(k \log k)$ time (and $\log(n/k)$ factor larger for Hadamard)
  - Comb filter: optimal when it works
  - Interval filter: $O(k \log n)$ samples and time

Approximately sparse: “optimal” is $O(k \log(n/k))$ samples and $O(k \log(n/k) \log n)$ time
  - Comb filter: optimal when it works
  - Interval filter: optimal samples OR optimal time OR $\log^c \log n$-competitive mixture.

Thank You